



VENTITY

**Entity-based
System Dynamics
modelling
Intro & Model Reuse**

#ISDC 2024

Logistics

- Download the course material folder and unzip to a convenient location
 - Source: <http://vensim.com/conference>
 - Choose a location you control, i.e. not locked by your operating system
- Install Ventity 5 from <http://Ventity.biz/download>
 - Licenses are unlocked through the end of August using **Bergen2024** as the license key
 - If you have an existing 4.5 or later, that should be fine too

Goals

Mechanics

- Environment
- Diagramming
- Equations
- Units
- Run control
- Run naming
- Entity initialization data
- Charts/tables
- Slider controls
- Entity picker
- Entity imports

Modeling Concepts

- Model-data separation
- Entities
- Attributes
- References
- Collections & aggregates
- Actions & triggers

Ventity concepts

- **Entitytype/Entity** – related structure sharing the same level of detail, often representing a sector or agent; analogous to classes and instances in OOP, or tables and rows in relational databases
- **Attribute** – a text tag that may contain a reference, or just categorical data
- **Reference** – a pointer to another entity, often defined by an attribute
- **Collection/Subcollection** – a set or subset of entities of a given type, containing **aggregates** for variable values of members
- **Action** – a discrete event between time steps that changes system structure or state, dispatched by a **trigger**

The screenshot shows the 'supplychainlevel' diagram in the software. The 'Attribute/Reference Grid' is visible, displaying a table with columns: Attribute Name, EntryReference, Is Key, Reference Name, EntryReference, and Condition. The table contains data for 'Level' and 'Supplier' attributes. Below the grid is the 'Diagram Toolbar' with various icons for diagram elements like Stock, Auxiliary, Link, Label, Notes, Chart, Button, Image, and Map-Sizer.

Attribute Name	EntryReference	Is Key	Reference Name	EntryReference	Condition
Level	supplychainlevel	YES	Supplier	supplychainlevel	attribute value
Supplier	supplychainlevel				attribute value

Orders

Entity	Min	supplychainlevel.supply line att...	Max	Input
retailer	0		2	1
wholesaler	0		2	1
distributor	0		2	1
factory	0		2	1

Entity	Min	supplychainlevel.stock adjustm...	Max	Input
retailer	2		8	4
wholesaler	2		8	4
distributor	2		8	4
factory	2		8	4

Inspector

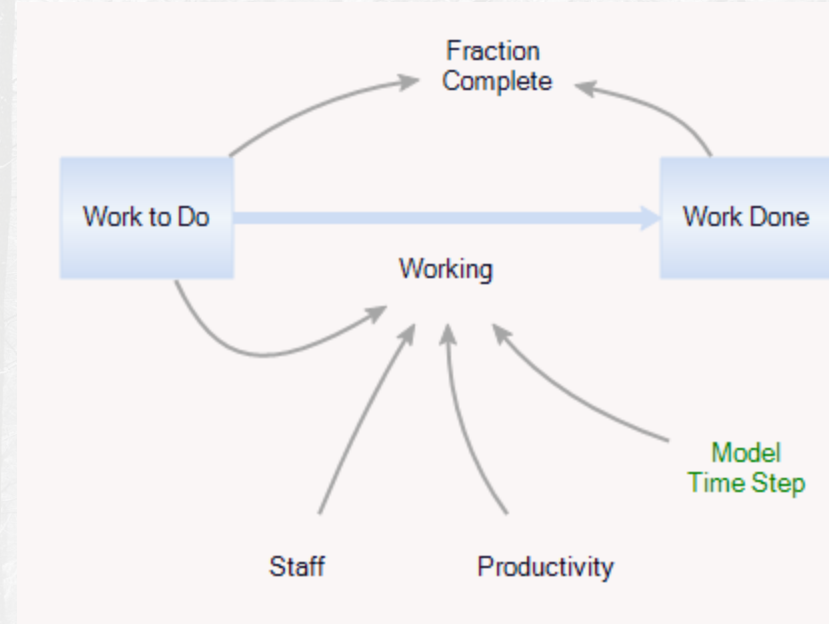
Console & Error List

Building & Running a Simple Model

- Project 0 – defining & running structure
- Project 1 – starting point for ...
- Project 2 – adding a prerequisite network
- Project 3 – adding entity initialization data
- ...
- Project 21 – greater realism

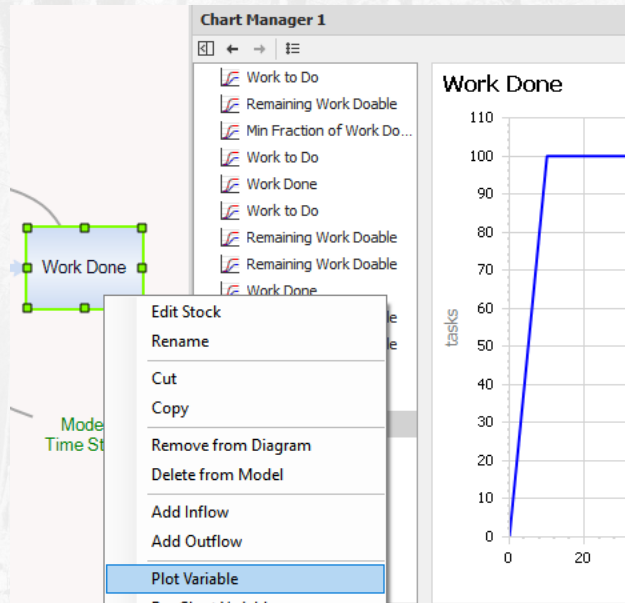
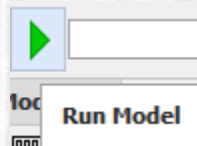
Project 0 - Building

- Create a new model
- Draw some structure
- Write the key equation
- $\text{Working} = \text{Min}(\text{Work to Do} / \text{Model.Time Step}, \text{Staff} * \text{Productivity})$



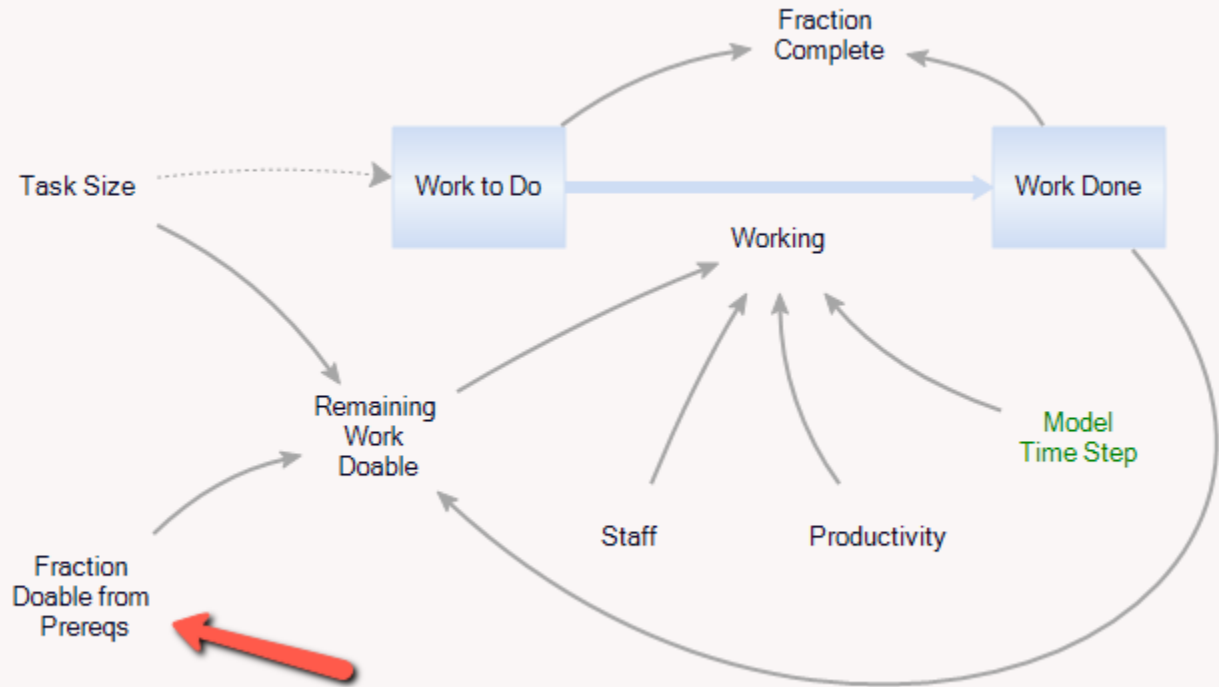
Project 0 – Running & Viewing

- Units
- Debugging
- Running
- Viewing



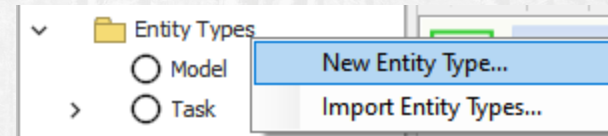
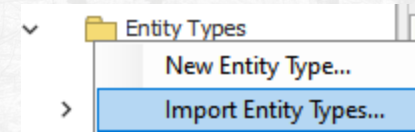
Project 1 – Starting Point

- Open the Project 1 model in the materials package
- Just take a look around

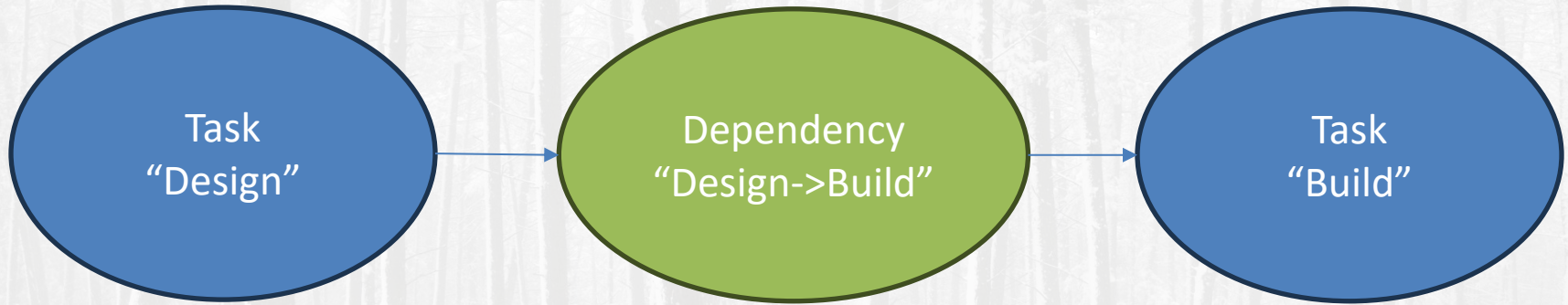


Project 2 - Reuse

- Create a new model
- Import the Task entity from Project 1
- Create a new Dependency entity

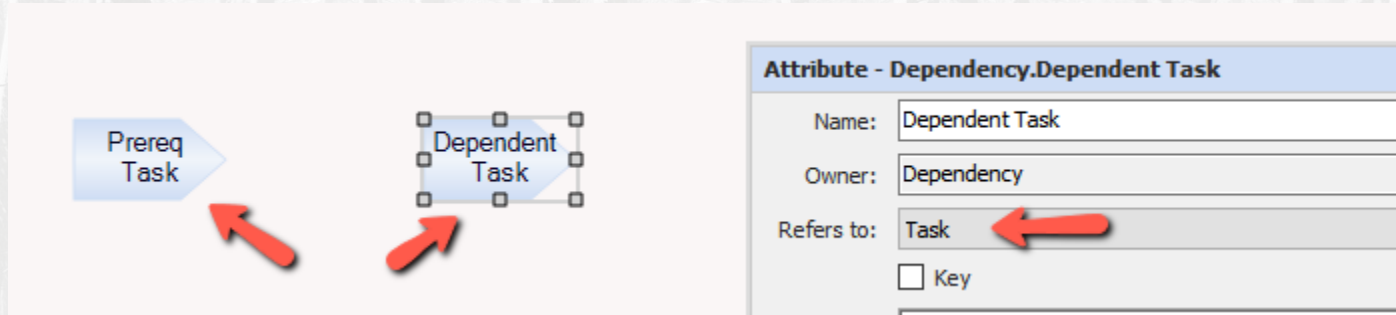


Project Architecture



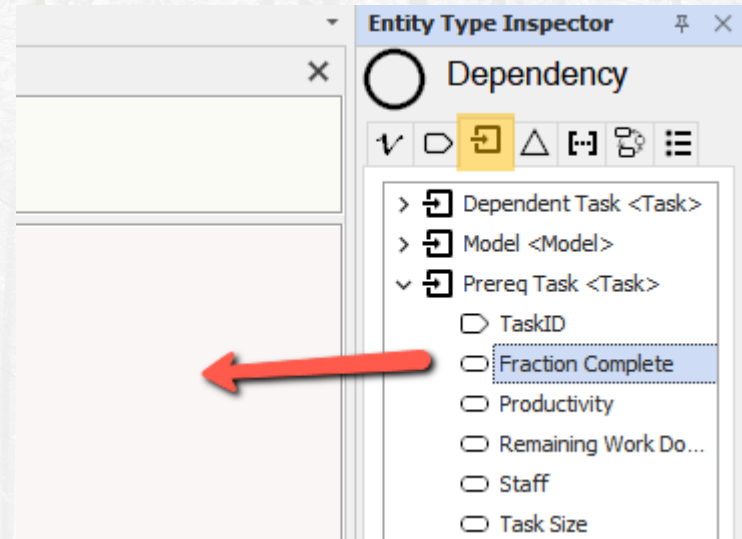
Project 2 – Add Prereqs

- In the Dependency entitytype, add 2 attributes and make them tasks



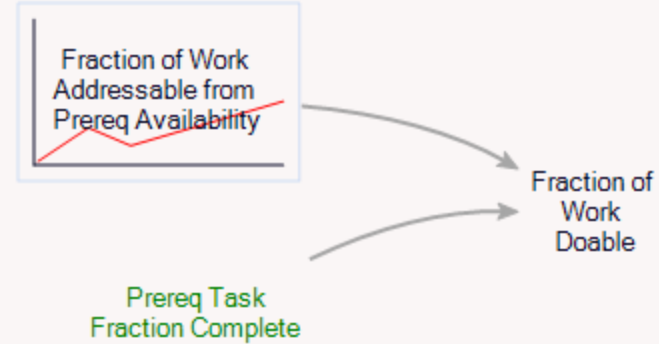
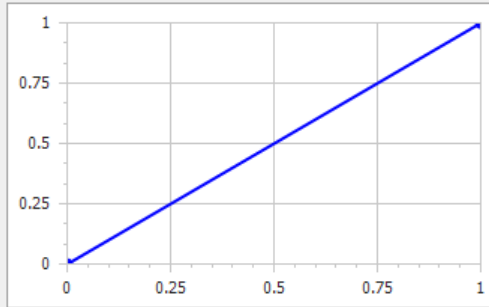
Project 2 – Referencing

- Open the inspector and switch to the Reference tab
- Expand the Prereq Task reference, and drag Fraction Complete onto the diagram



Project 2 - Lookup

- Add a lookup

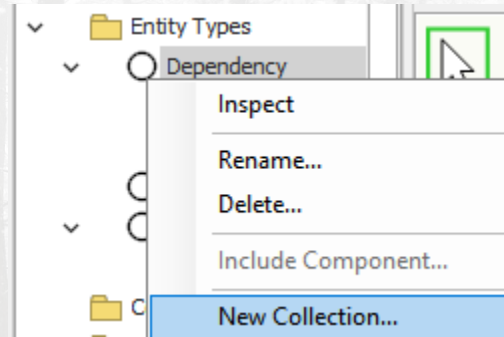


- Write an equation

Fraction of Work Doable = Fraction of Work Addressable from Prereq Availability(Prereq Task.Fraction Complete)

Project 2 - Subcollection

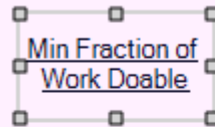
- Add a new (sub)Collection to the Dependency
- Add an aggregate Min(Fraction Doable)



New Collection of Dependency

Key Attributes:

<input type="checkbox"/>	DependencyID
<input checked="" type="checkbox"/>	Dependent Task
<input type="checkbox"/>	Prereq Task

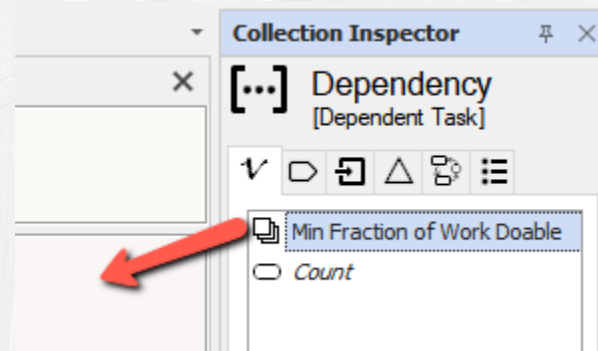


Aggregate - Dependency[Dependent Task].Min Fraction of Work Doable

Name:	Min Fraction of Work Doable
Owner:	Dependency[Dependent Task]
Units:	fraction
Base Type:	Dependency
Aggregation Function:	Min
Target Variable:	Fraction of Work Doable
If Empty Expression:	1

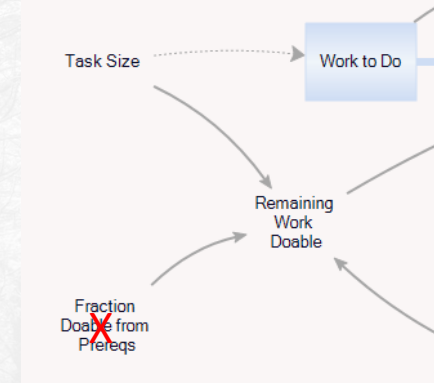
Project 2 – Referencing the Collection

- Return to the Task
- Drag the aggregate from the Dependency collection inspector or diagram, onto the Task diagram

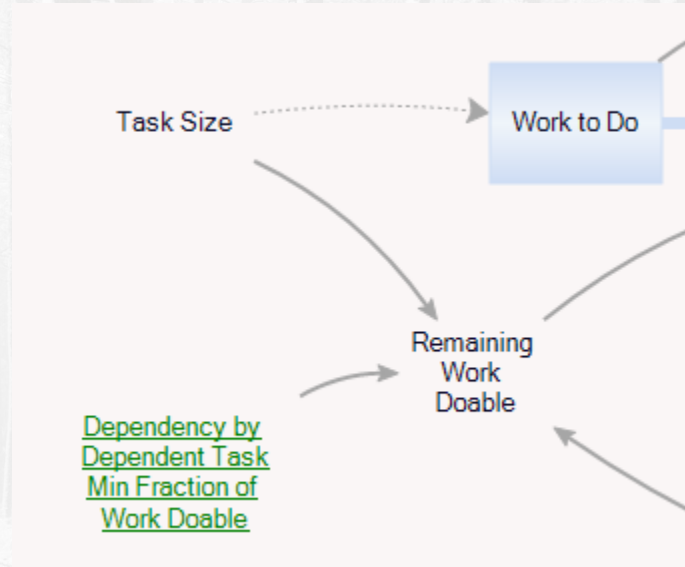


Project 2 - Connecting

- Delete the existing “Fraction Doable”



- Replace it with the referenced aggregate



Project 2 – Completing the Reference

- Point the reference to the TaskID

The screenshot shows a software interface with a context menu and a reference configuration dialog. The context menu is open over a node labeled "Dependency by Dependent Task". The menu options are: "Edit Aggregate", "Edit Reference 'Dependency by Dependent Task'", "Rename", "Show Primary Node", and "Show Base Variable". The "Edit Reference 'Dependency by Dependent Task'" option is selected. To the right, the "Reference - Task.Dependency by Dependent Task" dialog is open. It has fields for "Name" (Dependency by Dependent Task), "Owner" (Task), and "Target Type" (Dependency[Dependent Task]). Below these is a "Conditions" table with two columns: "Attribute Key" and "Value to Match". The table contains one row: "Dependent Task" and "TaskID". A red arrow points to the "TaskID" value.

Remaining Work Doable

Staff

Dependency by Dependent Task

Edit Aggregate

Edit Reference "Dependency by Dependent Task"

Rename

Show Primary Node

Show Base Variable

Reference - Task.Dependency by Dependent Task

Name: Dependency by Dependent Task

Owner: Task

Target Type: Dependency[Dependent Task]

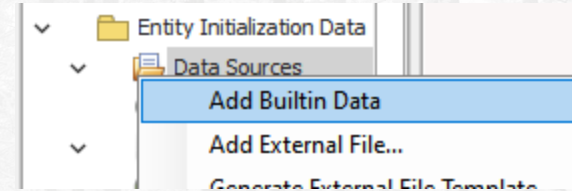
Conditions:

Attribute Key	Value to Match
Dependent Task	TaskID

- Run – expecting a few warnings

Project 3 – Populate the Data

- Create a new Builtin data source
- Add the Design & Build tasks



Dependency								Task
	<input checked="" type="checkbox"/>	Time	CalendarTime	TaskID	Staff	Work Done	Work to Do	
▶	<input checked="" type="checkbox"/>			Design	2			
2	<input checked="" type="checkbox"/>			Build	2			
★	<input type="checkbox"/>							

- Add the Design-Build dependency

Dependency								Task
	<input checked="" type="checkbox"/>	Time	CalendarTime	DependencyID	Dependent Task	Prereq Task	Fraction of Work Addressable from Prereq Availability	
▶	<input checked="" type="checkbox"/>				Build	Design	(0, 0), (0.5, 0), (0.8, 0.8), (1, 1)	

Project 21 – Greater Realism

- Take a look around
 - Collections
 - References
 - Model Map
- Modify the model
 - Modifying data
 - Split
 - Add
 - Data sets

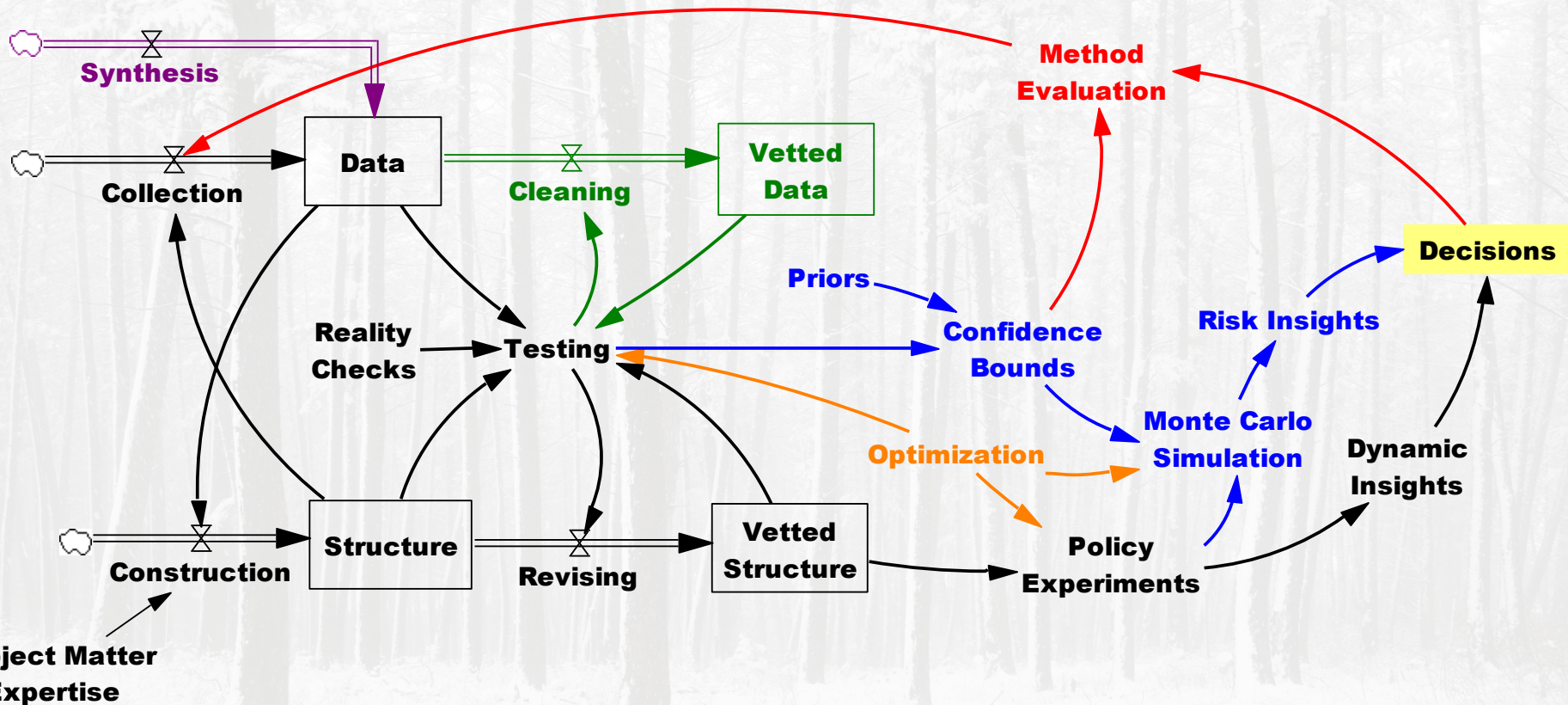


THANKS!



Our mission: Make good decisions by integrating all available information in models that are robust, transparent, and fast.

Dynamic Data Science



VENTITY helps teams collaborate to build complex models with model-data separation, modularity, dynamic creation of structure, sparse matrix relationships and intelligent agents
... and it's easy to use.

The approach lets you work on real problems, with real data, at a natural level of detail

Multilevel problems are ubiquitous

- Competitive dynamics, with new firms or products introduced midway through a simulation
- Payer/provider ecosystem, with patient migration and churn
- Doctor and patient behavior, with influence propagating on social networks or spatial grids
- Project model, with task prerequisite matrix loaded from data
- Supply chain with arbitrary organization
- Climate integrated assessment with physical and economic sectors delegated to different teams

Flexible aggregation options are needed

- Some problems *require* detail (can't aggregate in principle)
- ...But data availability may limit what is practical
- Too many dimensions make it hard to draw a stock-flow diagram
- Sometimes detail is easier (hard to decide a priori what aggregation is appropriate)
- Exploit Big Data and individual event statistics
- Show decision makers the granularity they live with (firms, brands, segments, regions, ...)

What do we need for productivity?

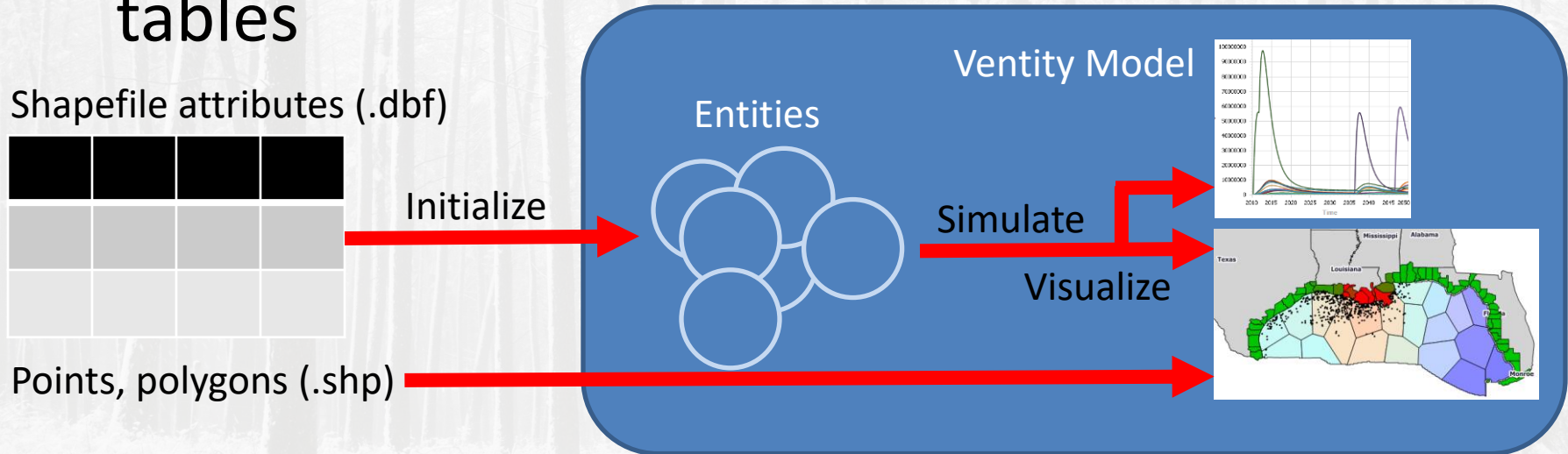
- Model-data separation, so model can be more generic
- Modularity:
 - Component reuse
 - Parallel development distributed among team members
 - Standalone calibration of submodels
- Sparse representation (not all drugs need to serve all indications)
- Structure change (addition of new brands)
- Easy data import, preprocessing and comparison with model output
- Fast, robust simulation engine
- Calibration optimization
- Attractive diagrams, output and user controls
- Calendar time instead of abstract months
- Good run management & archiving

Ventity concepts

- **Entitytype/Entity** – related structure sharing the same level of detail, like a class and instance in OOP, may represent a sector or agent
- **Attribute** – a text tag that may contain a reference, or just categorical data
- **Reference** – a pointer to another entity
- **Collection/Subcollection** – a set or subset of entities of a given type
- **Action** – a discrete event between time steps that changes system structure or state

Ventity + GIS

- Ventity's data model is a natural fit for relational data, including shapefile attribute tables



Which to choose?

Vensim if you need:

Interface running in browser client

MCMC

Command scripts

Matrix data

ODBC

RK integration

Ventity if you need:

Dynamic structure

Networks

Ad hoc data

GIS

If you don't need one of the central features available on one platform, it's largely a matter of taste.

If you're unsure, it's easier to port Vensim to Ventity.

Vensim vs. Ventity - Representation

Feature	Vensim	Ventity
Equation Editing	Menu Driven	Predictive Typing
Diagramming	Freeform	Constrained, Style sheets
Source Control	Improving with new diagram stability and modularity	Inherently friendlier due to modular entity structure
Data	Many complex paths, ODBC	A few simple paths, internal & Excel
GIS	Kludgy	Builtin
Sparse Matrices & Networks	Doable	Builtin
Dynamic Structure	No	Yes
Detail	Subscripts	Entity Collections

Vensim vs. Ventity - Connections

Feature	Vensim	Ventity
Internal Interfaces	Functional	Prettier
API	DLL, DDE	Windows Service
Web Deployment	Client (WebAssembly) or server (DLL/Linux library)	Server (Windows Service)
Data	Many paths, some complex	A few simple paths
GIS	Kludgy	Builtin
Algorithms	Optimization, MCMC, sensitivity, stochastic optimization	Basic optimization, sensitivity
Automation	Command scripts, DLL, DDE	Multi-run configuration, Windows Service

Ventity Advantages

- Reusability from modularity and model-data separation
- Natural relational data model
- Speed & clarity for sparse networks
- Flexibility of representation
- Elegant development environment

Porting Vensim -> Ventity

- Ventity can import a Vensim model
 - Subscripting is used to infer entitytypes (everything with similar dimensions is grouped)
 - Diagram must be reconstructed manually
 - Most equations are compatible, though a little hand editing may be needed
 - Some exotic functions like FIND ZERO don't have equivalents, or require a different structure (ALLOC functions)
- There's no automated return path to Vensim